The Most Probable Explanation for Probabilistic Logic Programs with Annotated Disjunctions

Dimitar Shterionov, Joris Renkens, Jonas Vlasselaer, Angelika Kimmig, Wannes Meert, Gerda Janssens

KULeuven, Belgium
(e-mail: firstname.lastname@cs.kuleuven.be)

Abstract. Probabilistic logic languages, such as ProbLog and CP-logic, are probabilistic generalizations of logic programming that allow one to model probability distributions over complex, structured domains. Their key probabilistic constructs are probabilistic facts and annotated disjunctions to represent binary and mutli-valued random variables, respectively. ProbLog allows the use of annotated disjunctions by translating them into probabilistic facts and rules. This encoding is tailored towards the task of computing the marginal probability of a query given evidence (MARG), but is not correct for the task of finding the most probable explanation (MPE) with important applications eg., diagnostics and scheduling.
In this work, we propose a new encoding of annotated disjunctions which allows correct MARG and MPE. We explore from both theoretical and experimental perspective the trade-off between the encoding suitable only for MARG inference and the newly proposed (general) approach.

Keywords: Probabilistic Logic Programming, Statistical Relational Learning, Most Probable Explanation, Logic Programs with Annotated Disjunctions, ProbLog

1 Introduction

The field of Probabilistic Logic Programming (PLP) combines probabilistic reasoning and logic to deal with complex relational domains. Most PLP techniques extend logic programming languages (such as Prolog) with probabilities. This has resulted in different probabilistic logic languages and systems, such as ProbLog [1], PRISM [2], LPADs [3], PHA [4] and others, which mostly focus on computing marginal probabilities of individual random variables or queries.

In contrast, the field of Statistical Relation Learning (SRL) often focuses on the extension of graphical models with logical and relational representations. Here, the most common inference tasks are computing the marginal probability of a set of random variables given evidence (MARG) and finding the most probable explanation given evidence (MPE).

The MPE task did not yet receive much attention in PLP, although it has important applications in diagnostics and scheduling. For a diagnostics example,

it is equivalent to finding the health state of a set of machines or components given a set of measurements. For a scheduling problem, one is interested in an optimal way of distributing resources or tasks. MPE is a special case of the computationally harder MAP (maximum a posteriori) task. The MPE is different from the task of finding the most probable proof in PLP, also called Viterbi proof [4–7], where one is interested in the proof to a query with highest probability.

ProbLog [1, 8] is a probabilistic logic and learning framework – a language and an engine. The engine handles various inference and learning tasks, including MPE inference [9], and as such tries to bridge the gap between PLP and SRL. Initially focused on probabilistic facts, or binary random variables, the ProbLog language now also supports Annotated Disjunctions (ADs) [3]. These specify exclusive choices between alternatives given some condition and thus provide an intuitive construct to represent multi-valued random variables. Internally, ProbLog encodes ADs by means of probabilistic facts and rules. However, given the semantics of ADs, this encoding is not correct for the MPE task.

In this paper, we propose an encoding of Annotated Disjunctions that is correct for both MARG and MPE inference. It uses ProbLog constraints, that is, first order logic sentences which restrict the models of ProbLog programs [10]. We compare the two encodings both theoretically and experimentally. We also incorporate an MPE inference algorithm based on weighted model counting into ProbLog.

The paper is organized as follows: In Section 2 we give background on the language ProbLog, introduce the MPE task and show why the encoding of ADs as ProbLog programs is not correct for the MPE task. Section 3 introduces our approach to encode ADs as weighted CNFs and proves its correctness. Section 4 summarizes experimental results. We conclude in Section 5.

2 Background

2.1 The ProbLog Language

The ProbLog language [1] is a probabilistic extension of Prolog. It uses probabilistic facts, that is, facts whose truth values are determined probabilistically to model basic uncertain data, and Prolog rules to define deterministic consequences of the probabilistic facts. A probabilistic fact has the form $p_i :: f_i$ and states that the fact f_i is true with probability $\mathit{true}(f_i) = p_i$, and false otherwise (with $\mathit{false}(f_i) = (1 - p_i)$). ProbLog uses the *distribution semantics* [11] to define a probability distribution over the least models (or *possible worlds*) of the ProbLog program. We write a possible world ω as a tuple (ω^+, ω^-), where the set ω^+ contains the probabilistic facts that are true and ω^- those that are false, thus, omitting the (uniquely determined) truth values for non-probabilistic atoms. Treating probabilistic facts as independent binary random variables, we obtain the probability of a possible world as:

$$P(\omega) = \prod_{f_i \in \omega^+} \mathit{true}(f_i) \prod_{f_i \in \omega^-} \mathit{false}(f_i) \tag{1}$$

Example 1. The following ProbLog program models a game with three bags with colored balls. One ball is picked from each bag, and the game is won if at least two balls are red. The probabilistic fact `0.6::red(b1)` states that the probability of selecting a red ball from bag `b1` is 0.6. It implies that selecting a ball with another color (eg. blue or green) has probability 0.4. The table outlines the set of its possible worlds (r_i abbreviates `red(bi)`) and their probabilities:

```
0.6::red(b1).
0.2::red(b2).
0.7::red(b3).
win:- red(b1), red(b2).
win:- red(b2), red(b3).
win:- red(b1), red(b3).
```

poss. world	r_1	r_2	r_3	win	P(r_1)	P(r_2)	P(r_3)	P(ω_i)
ω_1	T	T	T	T	0.6	0.2	0.7	0.084
ω_2	T	T	F	T	0.6	0.2	0.3	0.036
ω_3	T	F	T	T	0.6	0.8	0.7	0.336
ω_4	T	F	F	F	0.6	0.8	0.3	0.144
ω_5	F	T	T	T	0.4	0.2	0.7	0.056
ω_6	F	T	F	F	0.4	0.2	0.3	0.024
ω_7	F	F	T	F	0.4	0.8	0.7	0.224
ω_8	F	F	F	F	0.4	0.8	0.3	0.096

2.2 ProbLog Programs with Annotated Disjunctions

Annotated Disjunctions We recall annotated disjunctions as used in LPADs [3] and CP-Logic [12]. An annotated disjunction (AD) a is a multi-headed rule of the form $p_1 :: h_1; \ldots; p_n :: h_n \leftarrow b_1, \ldots, b_m.$, where p_i is the probability of the head atom h_i, the p_i sum to at most one, and the conjunction of the literals $b_1, \ldots, b_m$ forms the body of the AD. We denote with $head(a)$ the set of head atoms of the AD a and with $body(a)$ the set of body literals of a. If the body is true, the AD probabilistically *causes* one of the head atoms to become true, otherwise the AD does not have an effect. Thus, if the same atom appears in the heads of multiple ADs, they correspond to different causes. If $\sum_{i=1}^{n} p_i < 1$ there is a probability ($= 1 - \sum_{i=1}^{n} p_i$) that none of the head atoms is caused to be true. As this can be made explicit by adding an extra *none* head atom, we assume that probabilities sum to one.

We now summarize the process semantics of annotated disjunctions as defined in CP-logic; we refer to [12] for full technical details and a proof of the equivalence of this semantics with the instance based semantics of LPADs [3, 12].

Definition 1 (Probability Tree). *Let $A = \{a_1, \ldots, a_k\}$ be a set of ground annotated disjunctions over atoms L_A. A* probability tree *$T(A)$ is a tree where every node n is labeled with an interpretation $I(n)$ assigning truth values to a subset of L_A and a probability $P(n)$, constructed as follows:*

- *The root node $\bot$ has probability $P(\bot) = 1.0$ and interpretation $I(\bot) = \{\}$.*
- *Each inner node n is associated with an AD a_i such that*
  - *no ancestor of n is associated with a_i,*
  - *all positive literals in $body(a_i)$ are true in $I(n)$,*
  - *for each negative literal $\neg l$ in $body(a_i)$, the positive literal l cannot be made true starting from $I(n)$.*

  and has one child node for each atom $h_j \in head(a_i)$. The j^{th} child has interpretation $I(n) \cup \{h_j\}$ and probability $P(n) \cdot true(h_j)$.
- *No leaf can be associated with an AD following the rule above.*

The path from the root to a leaf n is called a selection σ_n *with probability* $P(\sigma_n) = P(n)$*. We say that each selection defines an interpretation and denote with* $I(\sigma_n)$ *the interpretation defined by* σ_n *(*$I(\sigma_n) = I(n)$*).*

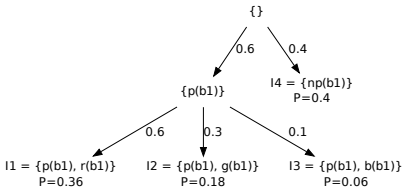
All probability trees for a given set of ADs define the same distribution over selections [12]; from here on, we refer to an arbitrary probability tree when mentioning "the probability tree of A".

Example 2. We consider a variant of the game in Example 1 with a single bag containing red, green and blue balls, as expressed by AD a_1, where the player randomly decides to pick a ball or not, as encoded by AD a_2:

a_1: `0.6::red(b1); 0.3::green(b1); 0.1::blue(b1) <- pick(b1).`

a_2: `0.6::pick(b1); 0.4::no_pick(b1) <- true.`

A probability tree associated with these ADs and its selections are:



Selection:	Interpretation	Probability $P(\sigma_i)$
σ_1	$I1 = \{$`pick(b1), red(b1)`$\}$	0.36
σ_2	$I2 = \{$`pick(b1), green(b1)`$\}$	0.18
σ_3	$I3 = \{$`pick(b1), blue(b1)`$\}$	0.06
σ_4	$I4 = \{$`no_pick(b1)`$\}$	0.4

Here, all selections define different interpretations.

ProbLog Encoding of Annotated Disjunctions To support MARG inference for ProbLog programs with annotated disjunctions, an AD is translated to a set of probabilistic facts with suitably adapted probabilities and a Prolog rule for each of its head atoms, with bodies which are mutually exclusive. We refer to [13, Chapter 3] for the details and illustrate the principle with an example.

Example 3. The ProbLog encoding for Example 2:

```
0.6::pf(1, 1).     0.75::pf(1, 2).
red(b1):- pick(b1), pf(1, 1).
green(b1):- pick(b1), pf(1, 2), \+ pf(1, 1).
blue(b1):- pick(b1), \+ pf(1, 2), \+ pf(1, 1).

0.6::pf(2, 1).
pick(b1):- pf(2, 1).
no_pick(b1):- \+ pf(2, 1).
```

2.3 Most Probable Explanations for ProbLog Programs

The inference task that has received most attention by the PLP community is computing the marginal probability of a ground query atom q given evidence $E = e$ on a subset of the other atoms, $\text{MARG}(q \mid E = e) = P(q \mid E = e)$. In the absence of evidence, this is also known as the success probability of a query.

In statistical relational learning (SRL) [14] and probabilistic graphical models [15] one of the key tasks is to find the most likely state of the world where a set of observations (the evidence) holds, also called MPE inference. Formally, the Most Probable Explanation (MPE) task, is defined as follows.

Definition 2 (Most Probable Explanation). *Given a probability distribution $P(V)$ over a set of discrete random variables V and a truth value assignment e to a subset of random variables $E \subseteq V$ (the evidence), the task of finding the most probable explanation (MPE) is to determine a truth value assignment u to the remaining random variables $U = V \setminus E$ with maximal probability, that is,* $\mathrm{MPE}(E) = \arg\max_u P(U = u \mid E = e)$.

A solution of the MPE task is also called an MPE state. Solution techniques developed for MPE include methods based on knowledge compilation [16], integer linear programming [17], and weighted MAX-SAT [18].

For a **ProbLog program without ADs**, V is the set of ground atoms, $E = e$ fixes truth values for a subset of those, and the task is to find the most likely assignment to all other atoms, that is, the possible world with the highest probability in which the evidence holds. As in [9], we assume ProbLog programs with finite groundings.

Example 4. For Example 1 given the evidence `red(b2)`=*true* the worlds where the evidence holds are ω_1, ω_2, ω_5 and ω_6. The one with the highest probability is ω_1. The MPE state is thus $\{\texttt{red(b1)} = \mathit{true}, \texttt{red(b2)} = \mathit{true}, \texttt{red(b3)} = \mathit{true}\}$.

For a set of **annotated disjunctions** $A = \{a_1, .., a_k\}$, the most probable explanation is the truth value assignment according to the interpretation defined by the selection in $T(A)$ with highest probability amongst the ones for which the evidence holds. Let $\mathcal{S}^{E=e} = \{\sigma | I(\sigma) \models E = e\}$ be the set of selections in which the evidence holds and $\hat{\sigma} = argmax_{\sigma \in \mathcal{S}^{E=e}} P(\sigma)$, then $MPE_A(E) = I(\hat{\sigma})$.

Example 5. For the set of ADs in Example 2, the evidence `blue(b1)` = *false* makes σ_3 invalid. The MPE state given the evidence thus is $\sigma_4 = \{\texttt{no\_pick(b1)}\}$.

The ProbLog encoding of these ADs (cf. Example 3) has the following possible worlds (`p(b1)` is short for `pick(b1)` and `np(b1)` for `no_pick(b1)`):

poss. world	`pf(1, 1)`	`pf(1, 2)`	`pf(2, 1)`	`red(b1)`	`green(b1)`	`blue(b1)`	`p(b1)`	`np(b1)`	$P(\omega_i)$
ω_1	T	T	T	T	F	F	T	F	0.27
ω_2	T	T	F	F	F	F	F	T	0.18
ω_3	T	F	T	T	F	F	T	F	0.09
ω_4	T	F	F	F	F	F	F	T	0.06
ω_5	F	T	T	F	T	F	T	F	0.18
ω_6	F	T	F	F	F	F	F	T	0.12
ω_7	F	F	T	F	F	T	T	F	0.06
ω_8	F	F	F	F	F	F	F	T	0.04

The evidence `blue(b1)` = *false* holds in all worlds except ω_7, and the MPE state of the ProbLog program is thus {`pf(1, 1)` = *true*, `pf(1, 2)` = *true*, `pf(2, 1)` = *true*}. For this MPE state `no_pick(b1)` is false and `pick(b1)` is true, which is different from the MPE state of the underlying set of ADs.

Example 5 illustrates that using the ProbLog encoding of annotated disjunctions can result in incorrect MPE states. The reason is that several possible worlds of the ProbLog encoding may correspond to the same selection in the probability tree. This is not a problem when computing marginal probabilities, as the probabilities of all possible worlds where the query is true are summed together.

In [9] MPE is mentioned as special case of the more general maximum a posteriori (MAP) inference. MAP is the task of finding the most likely values for a set of query atoms given *partial* evidence. The query atoms are a subset of all atoms of the ProbLog program for which evidence is not given. Solving then the MAP task requires to marginalize over the atoms which are neither given as queries nor as evidence. That is, we have to apply maximization over the query atoms given the evidence atoms and summation over the rest. MAP inference can be used in order to find the MPE state of a ProbLog program with ADs. For the ProbLog encoding of such a program we can give as queries all the atoms of that program excluding the probabilistic facts generated by the encoding. Then solving the MAP task will give the truth value assignments of these atoms which is in practice the MPE state of the initial program. The MAP inference is computationally expensive, while MPE can be computed efficiently [19, 16]. The new encoding we introduce in Section 3.1 allows to solve the MPE task on ProbLog programs with ADs both correctly and efficiently by enforcing a one-to-one correspondence between selections and possible worlds.

Relation to Most Probable Proof In PLP the term *most probable explanation* typically is used interchangeably with *most probable proof*, also called *Viterbi proof* [4–7]. A proof (or explanation) ω' for a query is a partial truth value assignment (or partial possible world) such that for all full assignments extending the proof, the query holds. Finding a most likely proof (the VIT task) is different from MPE in that it does not aim at finding the state of all unobserved variables, but an assignment to a small set of variables sufficient to explain a query. More formally, given a query q, we have $VIT(q) = \arg\max_{\omega' \in E(q)} P(\omega')$ with $E(q)$ the set of all explanations or proofs of q.

For certain types of models, finding the Viterbi proof for query q corresponds to solving MPE with $q = true$ as evidence. This is true for instance in programs modeling Hidden Markov Models, where both VIT and MPE have to make one choice per time point, but does not hold in general:

Example 6. In the program below, query `win` has two proofs: the first uses facts `red` and `green` and has probability $0.4 \cdot 0.9 = 0.36$, the second uses facts `blue` and `yellow` and has probability $0.5 \cdot 0.6 = 0.3$. The Viterbi proof for `win` is the first one. The MPE state for evidence `win` = *true*, however, is $\{\neg\texttt{red}, \texttt{green}, \texttt{blue}, \texttt{yellow}\}$. This state does not extend the Viterbi proof.

```
0.4::red.   0.9::green.        win :- red, green.
0.5::blue.   0.6::yellow.      win :- blue, yellow.
```

Relation to Most Probable Explanation for Bayesian Networks To encode a Bayesian network using Annotated Disjunctions, each row in each conditional probability table (CPT) is encoded as a deterministic rule to capture the value assignments and a probabilistic fact to capture the probability. There is only one CPT per variable (the child node) and the rows in a CPT express mutually exclusive value assignments to a set of variables (the *parents* of the node). For each value assignment to the parents we introduce one AD with head

atoms encoding the different value assignments to the child node and body associated to the value of the parents. The probability of each head atom is the probability given in the CPT for the value assignment of the parents. It has been shown that this encoding with ADs expresses the same probability distribution as the Bayesian network [20].

It can be shown that, given Definition 2, the MPE state of the Bayesian network is equivalent to the MPE state of a set of ADs that encode the Bayesian network. The probability tree inferred by a set of ADs that encode a Bayesian network has as property that each leaf has a unique interpretation. Therefore, the interpretation with the highest probability and the selection with the highest probability are equivalent. An intuitive proof can be constructed as follows: For any given CPT for a variable a, at each point in constructing the probability tree, if all parent variables of a certain CPT are present in the current partial interpretation, exactly one rule with a as head has a condition that is true. For each value of a a subtree is instantiated. None of the other rules can have a true condition in any of the subtrees due to the mutually exclusivity. This guarantees that each subtree has a different value assigned to a and no two interpretations in different subtrees can be identical.

3 Weighted CNF Encoding for Annotated Disjunctions

ProbLog inference is based on a transformation of the ProbLog program to a weighted Boolean formula in CNF, on which weighted model counting (WMC) is used to compute probabilities [8]. We now introduce a new encoding of annotated disjunctions in line with this transformation and consistent for MPE.

3.1 Encoding

The encoding of annotated disjunctions we propose (which we refer to as the *WMC* encoding to differentiate it from the *ProbLog* encoding discussed in Section 2.2.) has two parts: (i) a logic program with weighted facts that is transformed to CNF as in ProbLog; (ii) a set of constraints that is directly added to the weighted CNF. This is a special case of ProbLog with constraints (cProbLog) [10], adapted directly to the specific constraints needed here. ProbLog employs weighted model counting (WMC) techniques for MARG and MPE inference on the weighted CNF.

Definition 3 (WMC encoding for ADs). *The WMC encoding of a ground AD* $p_1 :: h_1; ..; p_n :: h_n \leftarrow b_1, .., b_m.$ *with unique identifier* $\mathtt{a_j}$ *consists of:*

- *for each* h_i *with* $1 \leq i \leq n$ *a* surrogate probabilistic fact $\mathtt{spf(a_j, h_i, i)}$ *with* $\mathit{true}(\mathtt{spf(a_j, h_i, i)}) = p_i$ *and* $\mathit{false}(\mathtt{spf(a_j, h_i, i)}) = 1.0$ *and a clause*

$$\mathtt{h_i} : -\mathtt{b_1}, .., \mathtt{b_m}, \mathtt{spf(a_j, h_i, i)}.$$

– *(the conjunction of) the following two constraints:*

$$\bigwedge_{i=1}^{n-1} \bigwedge_{l=i+1}^{n} (\neg \mathtt{spf}(\mathtt{a_j}, \mathtt{h_i}, \mathtt{i}) \vee \neg \mathtt{spf}(\mathtt{a_j}, \mathtt{h_l}, \mathtt{l})), \qquad \bigwedge_{k=1}^{m} \mathtt{b_k} \leftrightarrow \bigvee_{i=1}^{n} \mathtt{spf}(\mathtt{a_j}, \mathtt{h_i}, \mathtt{i})$$

Intuitively, surrogate probabilistic facts make the choices in ADs explicit, and constraints ensure that this does not introduce undesired combinations of values. When one head atom h_i is selected for an AD, the other head atoms are ignored. That is, they do not influence the probability of any selections with h_i true. That is why the false probability of a surrogate probabilistic fact is set to 1.0. The first constraint states that for each AD, at most one surrogate probabilistic fact can be true in any possible world, and thus only one head atom can be made true by the AD. The second constraint states that a choice is made if and only if the body of the AD is true. We write a surrogate probabilistic fact $\mathtt{spf}(\mathtt{a_j}, \mathtt{h_i}, \mathtt{i})$ with $\mathit{true}(\mathtt{spf}(\mathtt{a_j}, \mathtt{h_i}, \mathtt{i})) = p$ as $(p, 1.0) :: \mathtt{spf}(\mathtt{a_j}, \mathtt{h_i}, \mathtt{i})$.

Example 7. The WMC encoding of the two ADs of Example 2 consists of the program part:

```
(0.6, 1.0)::spf(1, red(b1), 1).
(0.3, 1.0)::spf(1, green(b1), 2).
(0.1, 1.0)::spf(1, blue(b1), 3).
red(b1):- pick(b1), spf(1, red(b1), 1).
green(b1):- pick(b1), spf(1, green(b1), 2).
blue(b1):- pick(b1), spf(1, blue(b1), 3).
(0.6, 1.0)::spf(2, pick(b1), 1).
(0.4, 1.0)::spf(2, no_pick(b1), 2).
pick(b1):- spf(2, pick(b1), 1).
no_pick(b1):- spf(2, no_pick(b1), 2).
```

and the four constraints where in the last one we omit the equivalence with *true*:

$$(\neg \mathtt{spf}(1, \mathtt{red(b1)}, 1) \vee \neg \mathtt{spf}(1, \mathtt{green(b1)}, 2)) \wedge (\neg \mathtt{spf}(1, \mathtt{red(b1)}, 1) \vee \neg \mathtt{spf}(1, \mathtt{blue(b1)}, 3)) \wedge (\neg \mathtt{spf}(1, \mathtt{green(b1)}, 2) \vee \neg \mathtt{spf}(1, \mathtt{blue(b1)}, 3))$$

$$\mathtt{pick(b1)} \leftrightarrow (\mathtt{spf}(1, \mathtt{red(b1)}, 1) \vee \mathtt{spf}(1, \mathtt{green(b1)}, 2) \vee \mathtt{spf}(1, \mathtt{blue(b1)}, 3))$$

$$\neg \mathtt{spf}(2, \mathtt{pick(b1)}, 1) \vee \neg \mathtt{spf}(2, \mathtt{no\_pick(b1)}, 2)$$

$$\mathtt{spf}(2, \mathtt{pick(b1)}, 1) \vee \mathtt{spf}(2, \mathtt{no\_pick(b1)}, 2)$$

The possible worlds of the program part in which the constraints hold are (`r`, `g`, `b`, `p`, `np` abbreviate `red(b1)`, `green(b1)`, `blue(b1)`, `pick(b1)`, `no_pick(b1)`):

poss. world	spf(1,r,1)	spf(1,g,2)	spf(1,b,3)	spf(2,p,1)	spf(2,np,2)	r	g	b	p	np	$P(\omega_i)$
ω_1	T	F	F	T	F	T	F	F	T	F	0.36
ω_2	F	T	F	T	F	F	T	F	T	F	0.18
ω_3	F	F	T	T	F	F	F	T	T	F	0.06
ω_4	F	F	F	F	T	F	F	F	F	T	0.40

3.2 Correctness

Theorem 1. *For a set $A = \{a_1, \ldots, a_k\}$ of ground annotated disjunctions, there is a bijection from the set $\mathcal{M}$ of models of the weighted CNF for A to the set $\mathcal{S}$ of selections in a probability tree T for A.*

Proof. *Let L_A be the set of atoms in A, and L_F the set of surrogate facts in the weighted CNF encoding of A. For every truth value assignment l_F to L_F, there is exactly one truth value assignment l_A to L_A such that $l_F \cup l_A$ is a model of the program part of the encoding [9]. The first constraint filters out all assignments l_F that assign true to more than one surrogate fact for the same ground AD,*

and the second filters out those that assign true to any surrogate fact for an AD whose body is false in $l_F \cup l_A$. Each remaining assignment $l_F \cup l_A$ is in one-to-one correspondence with a selection in $\mathcal{S}$. That is, each such an assignment (i.e. a model) corresponds to exactly one path from the root to a leaf of the tree T (the assignment l_A is a model of the node's interpretation) and there is one model for each path. ■

Theorem 2. *Given a set $A = \{a_1, \ldots, a_k\}$ of ground annotated disjunctions, the set $\mathcal{M}$ of models of the weighted CNF for A, and the set $\mathcal{S}$ of selections in a probability tree T for A, the weight of a model $M \in \mathcal{M}$ equals the probability of the corresponding selection $S \in \mathcal{S}$.*

Proof. *Follows directly from the fact that the model and the selection follow the same path through the tree and the definition of the weight function on the CNF. At each node, the probability of the selection so far is multiplied with the probability p_i of the chosen head atom, and the weight of the model with the weight of the AD's surrogate facts, $p_i \cdot 1.0 \cdot \ldots \cdot 1.0 = p_i$.* ■

Theorems 1 and 2 proof that our approach mirrors exactly the semantics of annotated disjunctions into ProbLog programs with FOL constraints. Example 8 illustrates the generality of our approach by encoding ADs which have the same atoms in their heads, i.e. the same event can result from multiple causes.

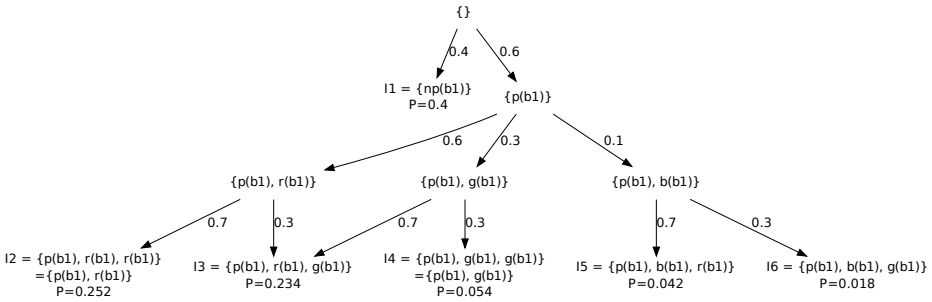
Example 8. Consider again the same problem as in the previous examples: a bag with colored balls. According to one source of information in the bag there are red, green and blue balls (as in Example 2); another source states that there are only red and green balls in the bag. This knowledge is expressed by the following program:

r_1: `0.6::red(b1); 0.3::green(b1); 0.1::blue(b1) <- pick(b1).`

r_2: `0.7::red(b1); 0.3::green(b1) <- pick(b1).`

r_3: `0.6::pick(b1); 0.4::no_pick(b1) <- true.`

The probability tree associated with these ADs is:



and the selections corresponding to the paths from the root to a leaf are:

Selection:	Interpretation	Probability $P(\sigma_i)$
σ_1	$I1 = \{$`no_pick(b1)`$\}$	0.4
σ_2	$I2 = \{$`pick(b1)`, `red(b1)`$\}$	0.252
σ_3	$I3 = \{$`pick(b1)`, `red(b1)`, `green(b1)`$\}$	0.108
σ_4	$I3 = \{$`pick(b1)`, `red(b1)`, `green(b1)`$\}$	0.126
σ_5	$I4 = \{$`pick(b1)`, `green(b1)`$\}$	0.054
σ_6	$I5 = \{$`pick(b1)`, `red(b1)`, `blue(b1)`$\}$	0.042
σ_7	$I6 = \{$`pick(b1)`, `green(b1)`, `blue(b1)`$\}$	0.018

Picking from the bag can result in selecting a red or green ball or red, green or blue ball (according to the different sources of information). In this case there are two selections (σ_2 and σ_3) which define the same interpretation ($I2$). When computing the MPE (as defined in Definition 2), each of these selections should be considered separately. The following table shows that our method corresponds exactly one possible world to each selections, regardless if they have the same interpretation and thus the MPE state can be computed correctly.

Possible World	pf(1, r)	pf(1, g)	pf(1, b)	pf(2, r)	pf(2, g)	pf(3, p)	pf(3, np)	r	g	b	p	np	Probability $P(\omega_i)$
ω_1	F	F	F	F	F	F	T	F	F	F	F	T	0.4
ω_2	T	F	F	T	F	T	F	T	F	F	T	F	0.252
ω_3	F	T	F	T	F	T	F	T	T	F	T	F	0.108
ω_4	F	F	T	T	F	T	F	T	F	T	T	F	0.126
ω_5	T	F	F	F	T	T	F	T	T	F	T	F	0.054
ω_6	F	T	F	F	T	T	F	F	T	F	T	F	0.042
ω_7	F	F	T	F	T	T	F	F	T	T	T	F	0.018

4 Evaluation

4.1 Analysis

The ProbLog pipeline for both MARG and MPE inferences consists of *Grounding*, *CNF conversion*, *sd-DNNF compilation* and *Evaluation* stages. Each previous stage produces input for the next one. ADs are processed in the Grounding stage. Using the ProbLog encoding the grounder generates only a ground program. On the other hand, invoking the WMC encoding results in (i) a ground program and (ii) CNF (φ_{ADs}) of the constraints to preserve the semantics of the ADs. In the second stage the ground program is converted to a (weighted) CNF formula φ_g. For the WMC encoding the CNF conversion generates the formula $\varphi = \varphi_g \wedge \varphi_{ADs} \wedge \varphi_E$, where φ_E is the CNF that states that the evidence (if any given) should hold (for the ProbLog encoding it is $\varphi = \varphi_g \wedge \varphi_E$).

For an AD with $|head| = n$ and $|body| = m$ the corresponding ProbLog encoding has n rules, with the n^{th} rule containing $m + n - 1$ facts in its body. The WMC encoding constructs rules with constant body size $(m + 1)$. Table 1 compares the number of variables and clauses in the CNF φ generated from the different encodings of an AD which does not introduce cycles. It shows that the ProbLog encoding produces smaller CNFs as compared to the WMC Encoding. Both encodings deal with each AD independently and define one (ground ProbLog) rule for each dependency between a head atom and the probabilistic facts generated by the encoding. The bodies of rules with the same head are combined in a disjunction in the CNF and introduce additional variables and clauses. For the two encodings of a set of ADs which share head atoms the number of additional variables and clauses in the CNF is the same. ProbLog employs

Encoding:	Number of CNF Variables:	Number of CNF Clauses:
ProbLog:	$2n + m - 1$	$\frac{n(2m+n+3)}{2} - 1$
WMC:	$2n + m$	$\frac{n(4m+n+3)}{2} + 1$

Table 1. Size of generated CNFs by the ProbLog and the WMC encoding.

the algorithm of [21] to break cycles. It has the same impact on both encodings. Table 1 ignores the additional clauses and variables in the cases where the ADs share head atoms and the ones introduced during loop breaking as they are the same for both encodings.

The sd-DNNF compilation stage is computationally the hardest in the pipeline of ProbLog. We use c2d [22] to compile a CNF into a sd-DNNF. c2d is non-deterministic (cf. [23]), that is, for the same CNF the resulting sd-DNNFs may differ. Hence, we cannot make theoretical estimations on the time for generating an sd-DNNF, its size or the time for its evaluation.

Our implementation of the evaluation procedure for the MPE task is according to [16, Chapter 12] and has the same complexity as the evaluation for the MARG task. Our experiments confirmed this claim.

4.2 Experimental Data

Our experiments aim to answer: **(i) What is the trade-off between the ProbLog encoding and the WMC encoding when doing MARG inference?**; **(ii) How does the WMC encoding scale w.r.t. the data size?**.

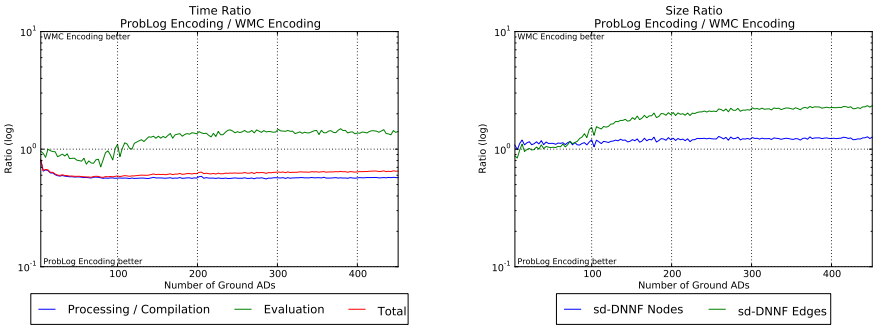
We analyze the results from experimenting on three artificially generated datasets. The first one (the *Balls* benchmark) is a more complex version of the ball game (cf. Example 2) that uses ADs to represent the different possible consequences of an action. The two other benchmarks are taken from [24]: we use the programs with annotated disjunctions with increasing head atoms (M_{gh}) and the ones with increasing number of (negated) body atoms (M_{gnb}). With the M_{gh} dataset we show the impact of the number of heads on the performance of the encodings. The M_{gnb} dataset shows the impact of the size of the bodies (that is, the second constraint of the WMC encoding). Our benchmarks can be found at `people.cs.kuleuven.be/~dimitar.shterionov/mpe`.

To assess the performance of our approach we measure the *processing-compilation* time (time for grounding plus CNF conversion plus sd-DNNF compilation), the evaluation time and the total inference time. We report the ratio $T^r = \frac{T(ProbLog)}{T(WMC)}$ per number of ground ADs which shows the relative time between the ProbLog and the WMC encoding. To assess the memory consumption we compare the number of nodes and edges for the generated sd-DNNF in each test run. We report the size ratio $S^r = \frac{S(ProbLog)}{S(WMC)}$.

Since c2d may generate different sd-DNNFs for the same CNF we run each test 5 times and report the average of time and size. We run our experiments on an 8-thread, Intel®Core™i7 @ 3400 MHz machine with 16GB RAM.

4.3 Experimental Results

Figure 1 summarizes the results for the *Balls* program. Figure 1 a) shows the time ratio T^r w.r.t. the number of ADs and Figure 1 b) the size ratio S^r.



a) b)

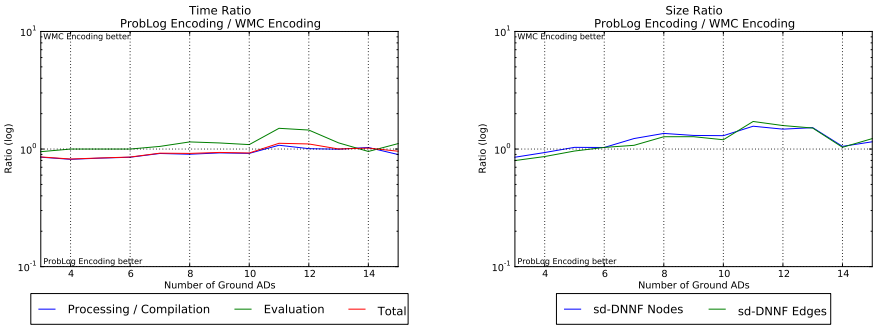
Fig. 1. Results from the *Balls* benchmark.

Figure 1 a) shows a worse processing-compilation time (blue line) for the WMC encoding compared to the ProbLog encoding but better evaluation time (green line). Since the processing-compilation time is much larger than the evaluation, the total time (red line) is also worse for the WMC encoding. Furthermore, the total time for the case of the WMC encoding is higher (as compared to the ProbLog encoding) with a constant factor over the number of ground ADs.

From Figure 1 b) we conclude that for almost all queries the compiled sd-DNNFs for the WMC encoding are smaller. This, together with the better evaluation time shows the WMC encoding to be preferable, since e.g., in diagnostics typically the model is compiled once and evaluated many times.

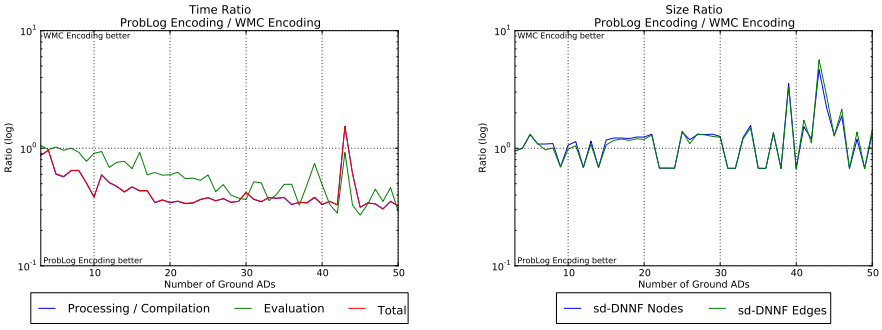
The results from experimenting on the M_{gh} dataset, summarized in Figure 2, show similar tendencies as the ones for the *Balls* benchmark. That is, the WMC encoding has worse processing-compilation and therefore also total execution time but still better evaluation time. Crucial for the encoding is the size of the body of the AD as it influences drastically the size of the CNF (cf. Table 1). Figure 3 shows how this affects the execution time. For the M_{gnb} dataset the WMC encoding results in worse processing-compilation, total and evaluation time than the ProbLog encoding.

Analyzing the results of our experiments we can state that there is a trade-off between the ProbLog encoding and the WMC encoding: generally, the total execution time for inference is worse (for the WMC encoding case). But the better evaluation time (*Balls* and M_{gh} benchmarks) due to the more compact sd-DNNFs makes our encoding preferable for problems where you compile once and evaluate many times. Also, in extreme cases, like the M_{gnb} benchmark, the WMC encoding does not perform better. Our experiments also show that with the two



a) b)

Fig. 2. Results from the M_{gh} benchmark.



a) b)

Fig. 3. Results from the M_{gnb} benchmark.

encodings ProbLog's inference scales in similar ways, with the ProbLog encoding outperforming the WMC encoding with respect to total execution time.

5 Conclusion

In this paper we introduced an encoding of annotated disjunctions as a weighted CNF formula (the *WMC Encoding*) to correctly reason with annotated disjunctions within the ProbLog system. ADs were supported previously by transforming them into probabilistic facts and Prolog rules. This approach (the *ProbLog encoding*) was correct for the MARG task but not for the MPE task. We showed that the WMC Encoding is correct both for the MARG and the MPE tasks. We then compared its performance to the ProbLog encoding with respect to MARG inference. Our new encoding is preferable for MPE inference and MARG inference with more than one evaluation (as it is the case for e.g. diagnostics

problems). While for the typical MARG task, the ProbLog encoding is better.

References

1. De Raedt, L., Kimmig, A., Toivonen, H.: ProbLog: a probabilistic Prolog and its application in link discovery. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence, AAAI Press (2007) 2468–2473
2. Sato, T., Kameya, Y.: Parameter learning of logic programs for symbolic-statistical modeling. Journal of Artificial Intelligence Research **15** (2001) 391–454
3. Vennekens, J., Verbaeten, S., Bruynooghe, M.: Logic programs with annotated disjunctions. In: Proceedings of International Conference on Logic Programming, Springer (2004) 431–445
4. Poole, D.: Logic programming, abduction and probability. New Generation Computing **11** (1993) 377–400
5. Sato, T., Kameya, Y.: A viterbi-like algorithm and EM learning for statistical abduction. In: Proceedings of Workshop on Fusion of Domain Knowledge with Data for Decision Support. (2000)
6. Kimmig, A., Demoen, B., De Raedt, L., Santos Costa, V., Rocha, R.: On the implementation of the probabilistic logic programming language ProbLog. Theory and Practice of Logic Programming **11** (2011) 235–262
7. Bragaglia, S., Riguzzi, F.: Approximate inference for logic programs with annotated disjunctions. In: Revised papers of ILP 2010. (2011) 30–37
8. Fierens, D., Van Den Broeck, G., Renkens, J., Shterionov, D., Gutmann, B., Thon, I., Janssens, G., De Raedt, L.: Inference and learning in probabilistic logic programs using weighted boolean formulas. Theory and Practice of Logic Programming (2014) accepted, http://arxiv.org/abs/1304.6810.
9. Fierens, D., Van den Broeck, G., Thon, I., Gutmann, B., De Raedt, L.: Inference in probabilistic logic programs using weighted CNF's. In: Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence. (2011) 211–220
10. Fierens, D., Van den Broeck, G., Bruynooghe, M., De Raedt, L.: Constraints for probabilistic logic programming. In Roy, D., Mansinghka, V., Goodman, N., eds.: Proceedings of the NIPS Probabilistic Programming Workshop. (2012)
11. Sato, T.: A statistical learning method for logic programs with distribution semantics. In: Proceedings of the 12th International Conference on Logic Programming (ICLP95), MIT Press (1995) 715–729
12. Vennekens, J., Denecker, M., Bruynooghe, M.: CP-logic: A language of causal probabilistic events and its relation to logic programming. Theory and Practice of Logic Programming **9** (2009) 245–308
13. Gutmann, B.: On Continuous Distributions and Parameter Estimation in Probabilistic Logic Programs. PhD thesis, KULeuven (2011)
14. Getoor, L., Taskar, B.: Introduction to Statistical Relational Learning (Adaptive Computation and Machine Learning). MIT Press (2007)
15. Koller, D., Friedman, N.: Probabilistic Graphical Models: Principles and Techniques. MIT Press (2009)
16. Darwiche, A.: Modeling and Reasoning with Bayesian Networks. Cambridge University Press (2009) Chapter 12.
17. Taskar, B., Guestrin, C., Koller, D.: Max-margin markov networks. In: Proceedings of Neural Information Processing Systems. (2003)

18. Richardson, M., Domingos, P.: Markov logic networks. Machine Learning **62**(1-2) (2006) 107–136
19. Huang, J.: Solving map exactly by searching on compiled arithmetic circuits. In: Proceedings of the 21st National Conference on Artificial Intelligence (AAAI-06. (2006) 143–148
20. Meert, W.: Inference and Learning for Directed Probabilistic Logic Models. PhD thesis, Informatics Section, Department of Computer Science, Faculty of Engineering Science (2011) Blockeel, Hendrik (supervisor).
21. Mantadelis, T., Janssens, G.: Dedicated tabling for a probabilistic setting. In Hermenegildo, M.V., Schaub, T., eds.: ICLP (Tech. Communications). Volume 7 of LIPIcs., Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2010) 124–133
22. Darwiche, A.: New advances in compiling CNF into decomposable negation normal form. In: Proceedings of the 16th European Conference on Artificial Intelligence. (2004) 328–332
23. Darwiche, A.: A compiler for deterministic, decomposable negation normal form. In Dechter, R., Sutton, R.S., eds.: AAAI/IAAI, AAAI Press/MIT Press (2002) 627–634
24. Meert, W., Struyf, J., Blockeel, H.: CP-logic theory inference with contextual variable elimination and comparison to BDD based inference methods. In: Proc. 19th International Conf. of Inductive Logic Programming. (2009) 96–109